AI Agentic & Orchestration Frameworks – Cheatsheet

This table summarizes major frameworks and tools for building agentic AI systems, multi-agent coordination, retrieval-augmented generation (RAG), and related capabilities:

Name	Type	Core Strength / Primary Use Case	Ideal For Scenarios	Language / Stack	Integration Style
LangChain	LLM orchestration framework (chains, agents, memory) (LangChain)	Highly modular toolkit for connecting LLMs with external tools, memory, and data sources. Provides prompt templates, memory stores, and "chains" to sequence model calls (LangChain). Enables rapid development of complex LLM-powered applications without reinventing common components.	Tool-using chatbots, QA systems with retrieval, conversational agents that require context memory, or any app chaining multiple LLM calls. Great for quick prototyping of LLM apps with varied modules (e.g. search, calculators).	Python (primary; also JS/TS, Java ports)	Composable APIs (classes for Chains, Agents, etc., assembled in code). Developers configure prompt templates and link predefined modules (retrievers, tools, memory) to form end-to-end workflows.
LangGraph	Agentic orchestration (graph-based, low-level control) (LangGraph)	Fine-grained control over multi-step and multi-agent workflows via a directed graph of nodes (stateful agents). More low-level and expressive than high-level agents – not tied to one "black-box" logic (LangGraph). Excels at building **controllable** and **reliable** agent flows (with explicit reasoning steps, error handling, etc.).	Complex or long-running tasks requiring explicit step-by-step control or multiple interacting agents. Ideal when you need to debug/monitor each reasoning step (e.g. a coding agent that requires rigorous reliability (LangGraph)) or implement custom decision logic beyond straightforward chains.	Python	Graph-defined workflows (developers create nodes for LLM calls, tool invocations, decision points, etc., and specify their connections). Stateful agent classes with persistent memory. Integration is code-driven (define and execute the graph), often alongside LangChain components (LangGraph).
CrewAI	Multi-agent orchestration framework (collaborative agents) ([CrewAI: A Guide With Examples of Multi AI Agent Systems	DataCamp](https://www.datacamp.com/tutorial/crew-ai#:~:text=CrewAI%20is%20a%20platform%20that,collaborate%20to%20perform%20complex%20tasks))	Open-source platform to build and deploy workflows where multiple AI agents with different roles collaborate on tasks ([CrewAI: A Guide With Examples of Multi AI Agent Systems	DataCamp] (https://www.datacamp.com/tutorial/crew-ai#:~:text=CrewAI%20is%20a%20platform%20that,collaborate%20to%20perform%20complex%20tasks)). Simplifies managing a "team" of agents (with shared memory and task delegation) to perform complex, multi-step objectives that one model alone might struggle with.	Scenarios requiring several specialized AI roles working together (e.g. a Researcher agent + a Writer agent for report generation). Useful for automating complex workflows by splitting them among agents (rather than a single monolithic chain). Particularly suited for projects that naturally decompose into sub-tasks handled by different experts.
AutoGen	Multi-agent conversation framework ([Multi-agent Conversation Framework	AutoGen 0.2](https://microsoft.github.io/autogen/0.2/docs/Use-Cases/agent_chat/#:~:text=AutoGen%20offers%20a%20unified%20multi,require%20using%20tools%20via%20code))	Enables multiple **conversable agents** (LLMs or humans + tools) to communicate via an automated chat interface to solve tasks collaboratively ([Multi-agent Conversation Framework	AutoGen 0.2] (https://microsoft.github.io/autogen/0.2/docs/Use-Cases/agent_chat/#:~:text=AutoGen%20offers%20a%20unified%20multi,require%20using%20tools%20via%20code)). Provides ready-made agent classes (e.g. AssistantAgent, UserProxyAgent) and manages the dialogue between them. Greatly simplifies orchestration of complex LLM workflows by having agents "talk" to each other to reason and act ([Multi-agent Conversation Framework	AutoGen 0.2](https://microsoft.github.io/autogen/0.2/docs/Use-Cases/agent_chat/#:~:text=abstraction%20of%20using%20foundation%20models,require%20using%20tools%20via%20code)).
OpenAgents (OpenAgents.ai)	Agent platform & SDK (open-protocol, multi-agent environment) (GitHub - xlang-ai/OpenAgents: [COLM 2024] OpenAgents: An Open Platform for Language Agents in the Wild)	An open platform for deploying and **hosting language agents** in real-world scenarios (GitHub - xlang-ai/OpenAgents: [COLM 2024] OpenAgents: An Open Platform for Language Agents in the Wild). Comes with a chat-based UI (agentic chat app) and a backend that supports open collaboration protocols (like MCP). Provides built-in agents (e.g. Data analysis agent, Plugin agent, Web browsing agent) that users can immediately utilize (GitHub - xlang-ai/OpenAgents: [COLM 2024] OpenAgents: An Open Platform for Language Agents in the Wild). Also offers a Python SDK ("Nodes") for developers to add custom agents or integrate the platform into apps.	Ideal if you want to *share or host* autonomous agents for others to use (similar to ChatGPT plugins but in an open ecosystem). Useful for quickly spinning up personal assistants, agents that can use web tools, or multi-agent services without building frontend/backend from scratch. Also a way to experiment with agent interoperability via standardized protocols in a community environment.	Frontend: TypeScript/React; Backend: Cloudflare stack (GitHub - OpenAgentsInc/openagents: The platform for AI agents. (wip)). Python SDK available for custom extensions.	Platform with web UI for agent chat, plus plugin-like integration. Agents can be composed through a graphical interface or via code using the SDK. Emphasizes **open protocols** (e.g. Nostr, MCP) for agent communication, enabling agents across different systems to talk to each other (GitHub - OpenAgentsInc/openagents: The platform for AI agents. (wip)). Integration involves registering agent "nodes" and specifying their tools/skills, which the platform then orchestrates in live chat or workflow.
DSPy	Declarative LLM orchestration framework ([What Is DSPy? How It Works, Use Cases, and Resources	DataCamp](https://www.datacamp.com/blog/dspy-introduction#:~:text=DSPy%20is%20an%20open,off%20prompting%20techniques))	Allows developers to build LLM pipelines using **modular, declarative programming** instead of hardcoded prompt scripts ([What Is DSPy? How It Works, Use Cases, and Resources	DataCamp] (https://www.datacamp.com/blog/dspy-introduction#:~:text=DSPy%20is%20an%20open,off%20prompting%20techniques)). You define what needs to be done (sequence of LLM tasks, error handling, optimization goals) and DSPy's engine (inspired by compiler principles) tunes prompts and manages flows. Particularly strong at self-optimizing pipelines: it can iteratively adjust prompts or weights to improve output quality.	Suitable for complex LLM applications where maintaining prompts and logic by hand would be tedious – e.g. building a robust RAG system with retry logic and evaluation, or an AI workflow that needs to optimize certain outputs (like ensuring factuality). Great for LLMops use cases: you can instrument and adjust your prompt chain declaratively and let the framework handle the rest.
LlamaIndex	Data-aware LLM framework (context augmentation / RAG) (LlamaIndex - LlamaIndex)	A leading framework for building LLM-powered applications **over your own data** (LlamaIndex - LlamaIndex). Offers tools to ingest and index documents (text, PDFs, SQL, etc.), then fetch relevant context to include in LLM prompts at query time (LlamaIndex - LlamaIndex). Simplifies creation of retrieval-augmented generation pipelines: you can quickly set up a knowledge base and ask questions with sources. Also supports advanced workflows (multi-step queries, combining multiple tools with an agent paradigm on top of the data).	Perfect for building QA bots and assistants that need to reference private or custom data (documents, knowledge bases). Use it when you want the LLM to have **up-to-date or domain-specific knowledge** beyond its base training – LlamaIndex handles the heavy lifting of connecting data to the LLM. Also useful for prototyping and production RAG systems (offers quick start for simple cases and flexibility for complex query workflows).	Python (core library; some support in TypeScript) (LlamaIndex - LlamaIndex)	Composable index/query modules. Typical use: ingest data into an index with a couple lines of code, then use a query engine to ask questions. Provides higher-level APIs (simple `query()` calls) as well as lower-level components to build custom pipelines. It also has an **agent tool interface** – indexed data can serve as a "tool" for an agent, allowing integration of retrieval with other actions in a sequence ([Understanding LlamaIndex: Features and Use Cases
Haystack	End-to-end **RAG & QA framework** (retrieval-augmented generation) (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex)	Powerful open-source framework for building production-ready search and QA systems on top of LLMs (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex). It provides a modular pipeline architecture: you compose **Nodes** (for tasks like document retrieval, generative answering, filtering) into a Pipeline that handles a user query (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex). Includes out-of-the-box support for popular vector databases, OpenAI/transformer models, and even a REST API to deploy your pipeline easily (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex).	Ideal for enterprise-grade Q&A systems, chatbots over documentation, or any scenario requiring reliable retrieval + generation. If you need to finely control each step (e.g., first retrieve top-10 docs, then refine, then feed to LLM), Haystack is well-suited. Also great when you want a ready-made **API endpoint** for your LLM-powered QA system – Haystack can serve pipelines as a service.	Python (with REST API; optional UIs) (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex)	**Pipeline-centric** integration. You can define pipelines declaratively (via Python code or YAML) by linking components. Each Node can be a prompt node, a tool, a reader model, etc. For complex needs, Haystack supports Agents which use LLMs to dynamically decide which tools/nodes to call in what order (Choosing a RAG Framework: Haystack, LangChain, LlamaIndex) (allowing more flexible

Name	Type	Core Strength / Primary Use Case	Ideal For Scenarios	Language / Stack	Integration Style
					flows), or you can stick to static DAG pipelines. In production, pipelines run in an API server or as part of your backend.
PydanticAI	Agent framework with **data validation** focus (PydanticAI)	A Python framework from the Pydantic team designed to bring rigorous **type checking and structured output** to LLM applications (PydanticAI). It acts as a shim between LLMs and Pydantic models – you define schemas for inputs/outputs and the agent ensures the LLM's responses conform to these schemas. Optimized for building **production-grade** apps, with features like robust error handling, instrumentation (via Pydantic's tooling), and integration with multiple model providers.	Great for use cases where you need the LLM to return JSON or well-formatted data that your program can rely on (e.g., form filling, code generation with specific classes, structured reports). Also ideal for enterprise settings: you can enforce contracts on what the AI produces, making it easier to trust and verify. If you liked FastAPI for web (data models + auto-validation), this brings a similar developer experience to AI agents (PydanticAI).	Python	**Pythonic class-based** integration. You define Pydantic models that represent your prompts and expected outputs, and write agent logic as standard Python functions/classes that use these models. PydanticAI wraps around LLM calls – it will handle calling the model, parsing the result into your Pydantic model (and retrying or error-correcting if validation fails). This means you use familiar Python control flow and data structures, with the framework invisibly managing the prompt formatting and validation behind the scenes.
AgentVerse	Multi-agent collaboration & simulation framework (AgentVerse - AI Agent)	Open-source framework for deploying multiple LLM-based agents that can either **collaboratively solve tasks or simulate environments** (AgentVerse - AI Agent). It provides two modes: (1) *Task-solving*, which assembles a team of agents to work together on a goal (e.g. coding co-pilots, brainstorming assistants), and (2) *Simulation*, which sets up a sandbox world where agents interact (useful for research on emergent behaviors or gaming scenarios) (AgentVerse - AI Agent). Comes with presets for common scenarios and logging tools to observe agent behavior.	Useful when a single agent is not enough – for complex problem-solving that benefits from multiple specialists (each agent can be given expertise). For example, an "Engineer" agent + "Manager" agent to collaboratively build software. Also a go-to framework for those researching **how multiple agents behave**: you can simulate social interactions, negotiations, or run multi-agent role-play experiments with ease. (It's been used for studying things like group decision-making, etc.) (AgentVerse - AI Agent)	Python (OpenBMB library)	**Scenario-based configuration**. Developers configure agents (roles, skills, and prompts) and define a task or environment in Python – then AgentVerse orchestrates the rest (running the agents in parallel turns, handling their communications). It includes utilities for things like shared memory or message boards in simulations. Integration might involve writing a script to set up the agents and environment parameters (or using provided case templates) and then invoking the framework's runner to execute the multi-agent session.
CAMEL	Multi-agent **role-playing** framework ([Revolutionizing AI Collaboration: Exploring the CAMEL Framework for Autonomous Multi-Agent Systems	by Vishnu Sivan	Feb, 2025	Medium](https://codemaker2016.medium.com/revolutionizing-ai-collaboration-exploring-the-camel-framework-for-autonomous-multi-agent-systems-26d2428020b7#:~:text=CAMEL%20AI%20,to%20achieve%20a%20common%20goal))	A framework that uses *role-playing* to enable autonomous agent cooperation. Each agent is assigned a specific role (e.g., AI User, AI Assistant, Task Planner) and they communicate in natural language to solve a task together ([Revolutionizing AI Collaboration: Exploring the CAMEL Framework for Autonomous Multi-Agent Systems
ReAct	Agent reasoning **paradigm** (Reason + Act prompting) (ReAct - Prompt Engineering Guide)	*Not a library, but an approach*: ReAct prompts an LLM to alternate between **thinking (reasoning)** and **acting (taking an action)** (ReAct - Prompt Engineering Guide). The core idea is that the model first produces a Chain-of-Thought explanation, then an action (like a tool use or query), then gets the result and repeats. This synergy lets the agent break down problems and interact with tools step-by-step, rather than one giant prompt. It's a foundational method behind many tool-using agents' success, as it systematically organizes the LLM's decision process.	Whenever an AI needs to perform multi-step reasoning or use tools/functions iteratively to reach an answer. For example, answering a complex question might require reasoning through sub-questions and performing searches/calculations – ReAct is ideal. It's commonly employed under the hood in frameworks like LangChain or Haystack Agents for tasks like web browsing agents, math problem solvers, etc. Essentially, use ReAct when a single-shot answer won't cut it and the model must **figure out a plan and execute actions** in loops.	N/A (technique is framework-agnostic; originally demonstrated via Python prompting)	**Prompt-as-code** style. To implement ReAct, one provides an LLM with a prompt template that encourages it to output reasoning thoughts and a proposed action (often formatted in a special way). The developer's code then parses that output: if an action is requested (e.g. "Search[query]"), the code executes it (e.g., performs the search) and feeds the result back into the model, appending to the prompt context. This loop continues until the model produces a final answer. Many agent frameworks incorporate ReAct by default – as a user you often just select a "ReAct agent" and the framework handles the prompt formatting and looping internally.
MCPs (Multi-Agent Collaboration Protocols)	Communication protocols/standards for multi-agent systems	Refers to emerging **protocols that structure how agents communicate and work together**. The idea is to define a common language or set of rules so that different agents (or agents on different platforms) can coordinate. For example, an MCP might specify message formats, role identifiers, turn-taking rules, or how to hand off tasks. This enables interoperability – multiple agents (and humans) can form a team following the protocol. (Anthropic's recent "Model Context Protocol" is one such attempt to standardize context sharing between AI models, and other efforts like SWARMS emphasize multi-agent protocol layers (Understanding SWARMS in One Article: Enterprise-level AI Agent …).)	Ideal when building systems with agents that are **modular or distributed**. If you want agents from different vendors or open-source projects to collaborate (or want a human and AI agent to communicate consistently), a collaboration protocol is key. Also important in decentralized or web-based agent networks (for instance, in Web3 or federated settings) where a **common protocol** lets agents discover and trust each other. In practice, using MCPs can lead to more robust multi-agent ecosystems, since each agent follows predictable interaction patterns.	N/A (conceptual standard; implemented via APIs, network calls)	**Protocol-driven integration**. When using an MCP, you will structure agent interactions according to that protocol – for example, exchanging JSON messages with specific fields (agent_id, content, performative, etc.) or following a workflow like request → deliberation → response. Some platforms (like OpenAgents or certain blockchain AI projects) provide libraries to handle MCP message passing. In essence, integration means your agent code conforms to the protocol (often by using a provided SDK or communication hub), allowing it to plug-and-play in a larger multi-agent system without custom integration for each new agent. (Understanding SWARMS in One Article: Enterprise-level AI Agent …)